# Relatório da rede de teste montada sob IPFS

Daniel Sousa <201605245> Tiago Carvalho <201605259>
Tiago Coelho <201604170>

## 1 Introdução

Como continuação da investigação desenvolvida em [1], no presente trabalho foi explorada uma tecnologia que implementa os princípios das ICNs, denominada IPFS [2]. Com base no utilitário GNS3 [3], foi montada uma topologia com diversas máquinas virtuais a funcionaram como *peers* da nossa rede (ou, mais adequadamente, do nosso enxame) IPFS, com vista a estudarmos a pilha protocolar que a suporta, ainda identificando algumas características que a tornam numa ICN.

## 2 Arquitetura e breve história do IPFS

O `(I)nter(P)lanetary (F)ile (S)ystem` é um sistema de ficheiros distribuído, i.e. *peer-to-peer*, que procura interligar todos os dispositivos constituintes de uma determinada rede no mesmo sistema de ficheiros. Esta tecnologia manifesta alguns paralelismos com a Web atual, porém, é mais correto interpretá-la como um único enxame BitTorrent [4]. Uma rede IPFS proporciona um modelo de armazenamento de conteúdo endereçado com alto débito, pois, ao contrário de uma arquitetura centralizada, as redes IPFS são construídas em torno de um sistema descentralizado de utilizadores que possuem uma parte dos dados gerais, removendo a existência de pontos únicos de falha. O conteúdo endereçado pelo IPFS origina um Merkle DAG, uma estrutura de dados segundo a qual é possível construir sistemas de ficheiros, *blockchains* ou até mesmo uma Web Permanente. Podemos interpretar o IPFS como uma Information Centric Network, que funciona como *overlay*; esta característica torna-o bastante resiliente, dado que resiste a ambientes opressivos de censura.

O IPFS é o produto do trabalho de Juan Benet, um jovem mexicano que, após a sua graduação em Computer Science pela Universidade de Stanford, decidiu fundar a sua própria empresa nos Estados Unidos, que seria comprada pela «Yahoo!» em 2013. Desde então, Benet fundou a Protocol Labs, a empresa que ainda hoje se dedica ao impulsionamento do projeto IPFS com o objetivo de eventualmente substituir protocolos desatualizados ainda em uso na Internet atual há mais de 20 anos.

Esta tecnologia intenta combater os problemas de segurança que lavram a Internet baseada em HTTP. O endereçamento e assinatura de conteúdo protegem os sites baseados em IPFS, tornando ataques DDoS impossíveis.

A humanidade está em rumo de uma realidade na qual o custo de entrega de conteúdo será maior do que os benefícios/lucros que esta entrega poderia providenciar, sendo que, atualmente, grandes empresas da Internet já se deparam com dificuldade em cumprir com as nossas exigências de conteúdo. Para colmatar estes problemas, empresas como a Google, Amazon e a Akamai empregam equipas dedicadas à sua resolução. Graças à rápida adesão do público a *smartphones* de baixo custo, continentes inteiros de consumidores migrarão para a realidade virtual conhecida com a Internet, na próxima década. Caso nada seja feito na medida da prevenção, a adesão de milhares de milhões de dispositivos com as suas

próprias exigências nos próximos anos poderão fazer com que a Internet não consiga servir todos os utilizadores que dela necessitam.

Todos os serviços com alta largura de banda que atualmente utilizamos benificiariam de uma Internet reconstruída sobre IPFS ou qualquer outra ICN. O seu desempenho iria melhorar dramaticamente graças à redução do custo e do tempo associados à entrega de conteúdo. Para além da QoS, as ICNs poderiam ajudar a Internet a crescer na direção do seu objetivo idealista, definido aquando da sua criação. Utilizando estes inovadores protocolos, o objetivo original, intrínseco da génese da Internet poderá ser realizado na sua totalidade: ser verdadeiramente capaz de conectar toda a população numa escala mundial.

# 3 Objetivos do trabalho desenvolvido

Neste trabalho laboratorial, vamos explorar as princípais características das ICNs identificadas em [1], verificando que são implementadas, ou minimamente replicáveis pelo IPFS. De modo a atingir esse objetivo, vamo-nos servir de vários exemplos práticos a fim de ilustrar o funcionamento das ICNs, e abordar algumas alternativas a funcionalidades que ainda não estão presentes no IPFS. Ao longo do nosso trabalho, vamos ainda fornecer uma nuance dos protocolos que constituem o IPFS, e o seu modo de operação.

# 4 Configuração da rede montada

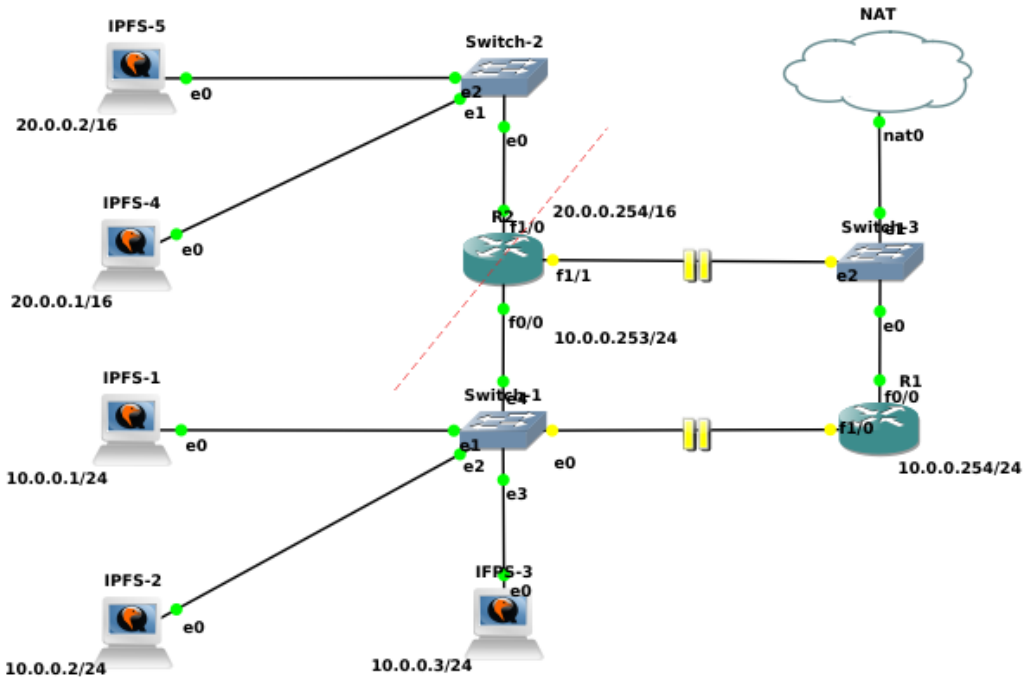


Figura 1: Topologia da rede montada

## 4.1 Sistema base das máquinas virtuais

Para cada máquina virtual, foram alocados 512 MiB de RAM e um núcleo de CPU. Foi carregada uma imagem de Debian [5], com o sistema base pré-istalado, proveniente do site OSBoxes [6], um processo que permitiu agilizar a configuração *boilerplate* dos sistemas.

As máquinas são executadas pelo emulador de processadores QEMU [7], o qual integra a tecnologia KVM [8] com vista a providenciar uma experiência aproximada do *hardware* real.

## 4.2 *Routers* utilizados

Foram emulados *routers* Cisco, como forma de:

- Definir um *default gateway* para cada uma das sub-redes montadas.
- Fazer NAT com o mundo exterior, permitindo acesso por SSH às máquinas virtuais, e descarregar algumas ferramentas utilizadas durante o processo de configuração.

## 4.3 Encaminhamento estático

Os IPs dos nós da rede e respetivas tabelas de encaminhamento foram configurados estaticamente, visto que a dimensão do enxame IPFS e o facto dos *peers* dependerem de um IP imutável para realizar o seu respetivo *bootstrap* não justificam empregar mecanismos de auto-configuração, tais como DHCP e OSPF.

### 4.3.1 Terminais Debian

Em cada terminal, o ficheiro `/etc/resolv.conf` foi configurado da seguinte forma:

```
nameserver 1.1.1.1
```

O servidor de DNS para o qual este IP aponta permite resolver os *hosts* de máquinas na Internet, as quais são usadas para obter utilitários necessários ao processo de configuração.

Na sub-rede `10.0.0.0/24`, as configurações relevantes, do ficheiro `/etc/network/interfaces`, são os seguintes:

```
allow-hotplug ens3
iface ens3 inet static
        # i é um dígito de 1 a 3,
        # correspondente a cada nó terminal da sub-rede
        address 10.0.0.i
        netmask 255.255.255.0
        gateway 10.0.0.254
        post-up ip route add 20.0.0.0/16 via 10.0.0.253
        pre-down ip route del 20.0.0.0/16 via 10.0.0.253
```

De modo semelhante, na sub-rede `20.0.0.0/16`, as configurações relevantes, do ficheiro `/etc/network/interfaces`, são os seguintes:

```
allow-hotplug ens3
iface ens3 inet static
        # i é um dígito de 1 a 2,
        # correspondente a cada nó terminal da sub-rede
```

```
address 20.0.0.i
netmask 255.255.0.0
gateway 20.0.0.254
post-up ip route add 10.0.0.0/24 via 20.0.0.254
pre-down ip route del 10.0.0.0/24 via 20.0.0.254
```

Desta forma, a rede `10.0.0.0/24` consegue comunicar com a rede `20.0.0.0/16`, e vice-versa.

#### 4.3.2 *Router* Cisco

Há dois *routers* configurados nesta rede, ambos servindo de *default gateway* à respetiva sub-rede a que pertencem, e desempenhando NAT com o mundo exterior. Consta-se ainda que o *router* R2 serve de *edge router* entre as duas sub-redes configuradas.

Segue-se a configuração dos routers pertinente ao encaminhamento estático:

```
Ri# conf t
Ri(config)# interface <interface interna à sub-rede>
Ri(config-if)# ip address <endereço IP estático do router> <máscara da sub-rede>
Ri(config-if)# end
Ri# copy run start
```

### 4.4 Configuração NAT

Embora não diretamente relacionada com a configuração do IPFS no nosso trabalho, por uma questão de completude, será aqui incluída a configuração do NAT nos *routers* Cisco:

```
Ri# conf t
Ri(config)# interface <interface externa>
Ri(config-if)# ip address dhcp
Ri(config-if)# ip nat outside
Ri(config-if)# exit
Ri(config)# interface <interface interna>
Ri(config-if)# ip nat inside
Ri(config-if)# exit
Ri(config)# ip nat inside source list 1 interface <interface externa> overload
Ri(config)# access-list 1 permit any
Ri(config)# ip nat inside source static tcp 10.0.0.j 22 interface <interface externa> 100j
Ri(config)# end
Ri# copy run start
```

### 4.5 Configuração do IPFS

A configuração dos nós terminais Debian que visa criar uma rede *overlay* IPFS será descrita nos próximos passos.

Começámos por obter uma distribuição do IPFS de [9], num dos nós, que posteriormente foi replicada por `scp` nos restantes terminais. Após a instalação do binário obtido no `$PATH`, seguiram-se os comandos:

```
# inicia as configurações por defeito do IPFS,
# no diretório ~/.ipfs
$ ipfs init
```

```
# como o objetivo é formar uma rede privada, vamos
# remover os nós bootstrap da configuração default
$ ipfs bootstrap rm --all

# vamos agora configurar os endereços de escuta do daemon IPFS,
# para estarem disponíveis em todos os nós da nossa rede
$ ipfs config Addresses.API /ip4/0.0.0.0/5001
$ ipfs config Addresses.Gateway /ip4/0.0.0.0/8080
```

Estamos quase prontos para iniciar o `daemon` do IPFS; falta agora gerar uma chave para o nosso enxame privado, para que evite interagir com o enxame público do IPFS. Para tal, vamos usar o seguinte programa do Go [10]:

```
package main

import (
    "fmt"
    "crypto/rand"
    "encoding/hex"
)

func main() {
    key := make([]byte, 32)
    _, err := rand.Read(key)
    if err != nil {
        panic(err)
    }
    fmt.Println("/key/swarm/psk/1.0.0/")
    fmt.Println("/base16/")
    fmt.Print(hex.EncodeToString(key))
}
```

O output deste programa será guardado no local `~/.ipfs/swarm.key` de cada nó constituinte da nossa rede. Vamos agora criar um serviço do `systemd` [11] para o `daemon` do IPFS:

```
$ cat /etc/systemd/system/ipfsd.service
[Unit]
Description=IPFS Daemon
After=syslog.target network.target remote-fs.target nss-lookup.target

[Service]
Type=simple
ExecStart=/usr/local/bin/ipfs daemon --enable-namesys-pubsub
User=root

[Install]
WantedBy=multi-user.target
```

Depois de iniciar o `daemon`, vamos agora identificar o nó que fará o `bootstrap` da nossa rede:

```
# este comando será usado para obter o ID do nó 10.0.0.1 na rede IPFS
$ ipfs id
{
        "ID": "QmdWpNesubhCkCJ3gm5jQcJWSZxpLfGmt8tQARZsk4UXrD",
```

```
        "PublicKey": ...,
        "Addresses": [
                "/ip6/::1/tcp/4001/ipfs/QmdWpNesubhCkCJ3gm5jQcJWSZxpLfGmt8tQARZsk4UXrD",
                "/ip4/127.0.0.1/tcp/4001/ipfs/QmdWpNesubhCkCJ3gm5jQcJWSZxpLfGmt8tQARZsk4UXrD",
                "/ip4/10.0.0.1/tcp/4001/ipfs/QmdWpNesubhCkCJ3gm5jQcJWSZxpLfGmt8tQARZsk4UXrD"
        ],
        "AgentVersion": "go-ipfs/0.4.22/",
        "ProtocolVersion": "ipfs/0.1.0"
}

# será utilizado o nó 10.0.0.1 como bootstrap em todos
# os nós, menos no próprio
$ ipfs bootstrap add /ip4/10.0.0.1/tcp/4001/ipfs/QmdWpNesubhCkCJ3gm5jQcJWSZxpLfGmt8tQARZsk4UXrD
```

Momentos depois, verificamos que no nó `10.0.0.1` se obtém o seguinte resultado:

```
$ ipfs swarm peers
/ip4/10.0.0.2/tcp/4001/ipfs/QmZTpyUUMVJtS4rewPgovoWTiFZUqKhb4eVjkwad42ayrS
/ip4/10.0.0.3/tcp/4001/ipfs/QmQXSpwQ23pZhuidjcxNTsFWAFz4KM8tdk4a97NPbFFrJJ
/ip4/20.0.0.1/tcp/4001/ipfs/QmUqHbqUSxipHs29nPzwjc4M7EQqDrYqZrw4mpmfzWnoBb
/ip4/20.0.0.2/tcp/4001/ipfs/QmRPjvZwfikrt3hG6DvYftPxBD5QwL61Vd1nBkZceZAEjU
```

Foi ainda desenvolvido outro programa, tomando partido do sub-sistema de `publish-subscribe` da implementação que estamos a usar do IPFS, com o intuito de fazer *cache* de conteúdo num determinado nó quando este é pedido por outro nó terminal:

```
package main

import (
    "os"
    "log"
    "os/signal"
    "syscall"
    "sync"

    ipfs "github.com/ipfs/go-ipfs-api"
)

const (
    localAddr   = "localhost:5001"
    pubSubTopic = "cache-cid"
)

var wg sync.WaitGroup

func main() {
    // open new connection to IPFS API
    sh := ipfs.NewShell(localAddr)

    // listen to IPFS nodes who advertise
    // the availability of some content
    sub, err := sh.PubSubSubscribe(pubSubTopic)
    if err != nil {
        panic(err)
```

```
    }

    // listen to os signals to end execution
    sigs := make(chan os.Signal)
    signal.Notify(sigs, syscall.SIGINT, syscall.SIGHUP, syscall.SIGTERM, syscall.SIGQUIT)
    go func() {
        <-sigs
        sub.Cancel()
        wg.Wait()
        os.Exit(0)
    }()

    for {
        // retrieve new content id
        msg, err := sub.Next()
        if err != nil {
            continue
        }

        // cache the content
        wg.Add(1)
        go cacheCID(sh, msg)
    }
}

func cacheCID(sh *ipfs.Shell, msg *ipfs.Message) {
    cid := string(msg.Data)
    err := sh.DagGet(cid, nil)
    if err == nil {
        log.Printf("Got %q from %q\n", msg.Data, msg.From)
    }
    wg.Done()
}
```

# 5 Resultados

Nesta secção, pretendemos demonstrar alguns exemplos do funcionamento da rede IPFS montada no âmbito deste trabalho, de modo a ilustrar e clarificar o porquê de certas decisões tomadas relativamente à arquitetura da nossa topologia da rede.

Em todos estes exemplos, os pacotes de *payload* IPFS vão ser diferenciados através do seu próprio tamanho, uma vez que o seu conteúdo está cifrado com a `libp2p`.

## 5.1 Exemplo 1 – *Lazy retrieval*

Nesta instância, será testado o modo como os nós IPFS fazem *cache* do conteúdo. Verificamos experimentalmente que os dados apenas são adicionados à *cache* no momento em que são adicionados por um nó à rede IPFS, ou obtidos via um comando `get` ou `cat`.

Foi criado um ficheiro `test_a.bin`, o qual foi adicionado à máquina `IPFS-1`, resultando numa *hash* única que o identificará e permitirá a sua posterior remoção.

Após executarmos um comando `get`, confirmámos que de facto o ficheiro adicionado se encontrava disponível na rede IPFS. Para o remover da *cache*, executamos o comando `ipfs pin rm`, que remove o `pin` do ficheiro, o qual evitava este ser removido por uma *garbage collection sweep* (i.e. limpeza da *cache*), que pode ser manualmente despoletada com o comando `ipfs repo gc`.

Uma vez apagado este ficheiro da *cache*, verificamos que é impossível obter esse conteúdo ao efetuar pedidos `ipfs get` utilizando a *hash* gerada anteriormente, ao expirar o `timeout` de 3 segundos que disponibilizámos ao comando invocado.

Concluímos assim que os ficheiros só são replicados na *cache* dos nós no momento em que um pedido é efetuado, poupando em largura de banda.

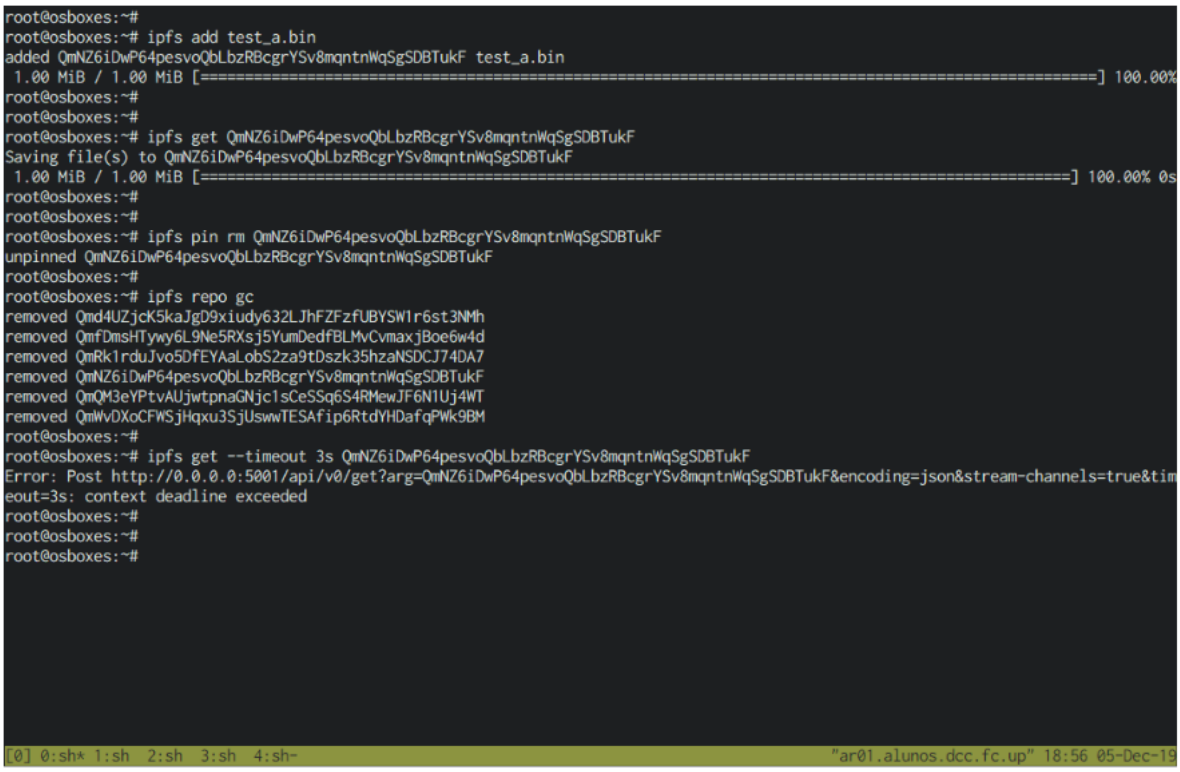


Figura 2: Exemplo 1

## 5.2 Exemplo 2 – Transferência do conteúdo por identificador

No exemplo que se segue, testaremos o processo simples de transferência dos dados por identificador único entre duas máquinas.

Numa fase inicial, foi gerado um ficheiro `test_b.bin` de cerca de 1 MiB, o qual foi adicionado à máquina `IPFS-1`, como ilustrado na Figura 3.

Em seguida, na maquina `IPFS-4`, é efetuado um pedido `get` do conteúdo, usando a *hash* gerada anteriormente como argumento, armazenando assim o conteúdo na sua *cache*.

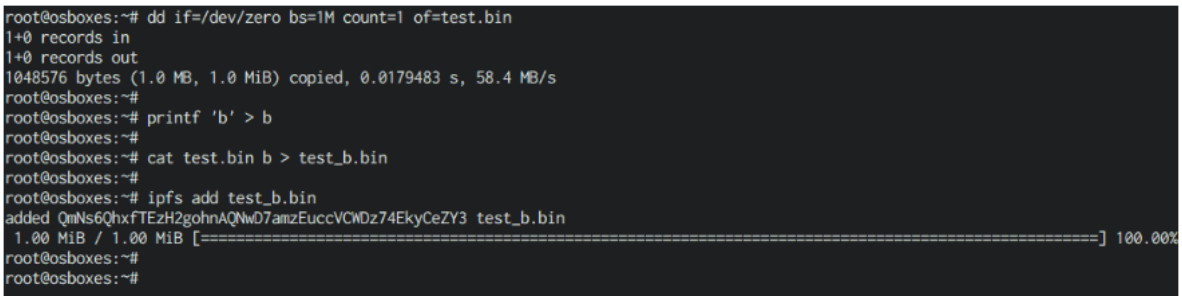


Figura 3: Exemplo 2, parte 1

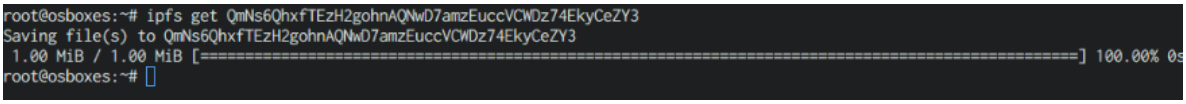


Figura 4: Exemplo 2, parte 2

Finalmente, através da ferramenta Wireshark, conseguimos filtrar e analisar a transmissão da sequência de pacotes entre as máquinas `IPFS-1` e `IPFS-4` (com os respetivos IPs `10.0.0.1` e `20.0.0.1`).

Como não é possível decifrar o conteúdo de cada pacote, é feita a sua distinção a partir da análise do tamanho de cada um, com o seguinte filtro do Wireshark:

```
ip.dst == 20.0.0.1 and tcp.payload and tcp.port == 4001 and tcp.len > 300
```

O conteúdo fica então disponivel nas máquinas `IPFS-1` e `IPFS-4`.

## 5.3 Exemplo 3 – *Caching*

Na instância seguinte, para melhor demonstrar a funcionalidade de *caching* do do IPFS na rede montada, procedemos à remoção do conteúdo da primeira máquina, `IPFS-1`, novamente via comandos `ipfs pin rm [hash]` e `ipfs repo gc`, como está ilustrado na Figura 6.

Posteriormente, na máquina `IPFS-5`, é invocado o pedido `ipfs get` com o *content id* do ficheiro, obtendo os dados que então apenas existiam na máquina `IPFS-4`, como ilustrado na Figura 7.

Por fim, podemos constatar, através da Figura 8, que, de facto, a sequência de pacotes é transmitida da máquina `IPFS-4` para a máquina `IPFS-5`, após a invocação do pedido `get`.

Assim, ambas as máquinas `IPFS-4` e `IPFS-5` ficam com o conteúdo de dados à disposição na sua *cache*.

## 5.4 Exemplo 4 – Transmissão do conteúdo a partir de dois *hosts*

Neste exemplo, pretendemos analisar como é realizada a recolha do conteúdo quando é realizado um pedido `get` ou `cat` e os respetivos dados estão presentes em mais do que uma máquina.

Para tal, na terminal da máquina `IPFS-2`, efetuamos a chamada do comando `get` com a *hash* gerada no segundo exemplo dado, com o intuito de guardar os dados recebidos em *cache*, como ilustrado na Figura 9.

Na captura Wireshark da Figura 10, podemos claramente verificar que existe uma distribuição de carga na transmissão dos pacotes de dados entre as duas máquinas que contêm o conteúdo, `IPFS-4` e `IPFS-5`. Esta distribuição é eficiente, visto que permite que pacotes com elevado tamanho sejam transmitidos

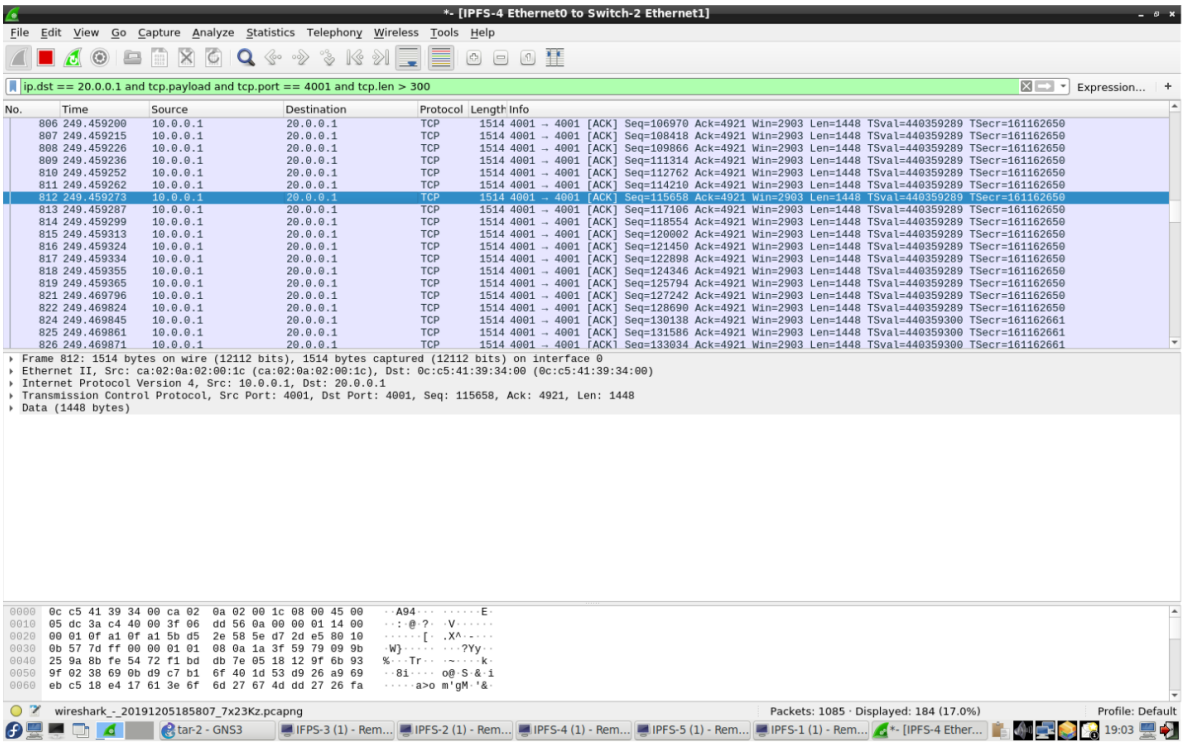


Figura 5: Exemplo 2, parte 3

```
root@osboxes:~# ipfs pin rm QmNs6QhxfTEzH2gohnAQNwD7amzEuccVCWDz74EkyCeZY3
unpinned QmNs6QhxfTEzH2gohnAQNwD7amzEuccVCWDz74EkyCeZY3
root@osboxes:~#
root@osboxes:~# ipfs repo gc
removed QmNs6QhxfTEzH2gohnAQNwD7amzEuccVCWDz74EkyCeZY3
removed QmQqNFJJTfTtPqdVpdgdYYDnkgafr7kDpng2FSQwXYGB4E
removed QmRk1rduJvo5DfEYAaLobS2za9tDszk35hzaNSDCJ74DA7
removed QmQLd9KEkw5eLKfr9VwfthiWbuqa9LXhRchWqD4kRPPWEf
removed QmUDCgcXcfKf4zgsVBWgMwqkMyNrZUYwioq8rdcgR1HMFs
removed QmRektwnQYtqs2gQs6Mw5Pkyu936Ls9B9sS8259keruSwB
root@osboxes:~#
root@osboxes:~#
root@osboxes:~#
```

Figura 6: Exemplo 3, parte 1

```
root@osboxes:~# ipfs get QmNs6QhxfTEzH2gohnAQNwD7amzEuccVCWDz74EkyCeZY3
Saving file(s) to QmNs6QhxfTEzH2gohnAQNwD7amzEuccVCWDz74EkyCeZY3
 1.00 MiB / 1.00 MiB [=============================================================================================] 100.00% 0s
root@osboxes:~#
root@osboxes:~#
```

Figura 7: Exemplo 3, parte 2

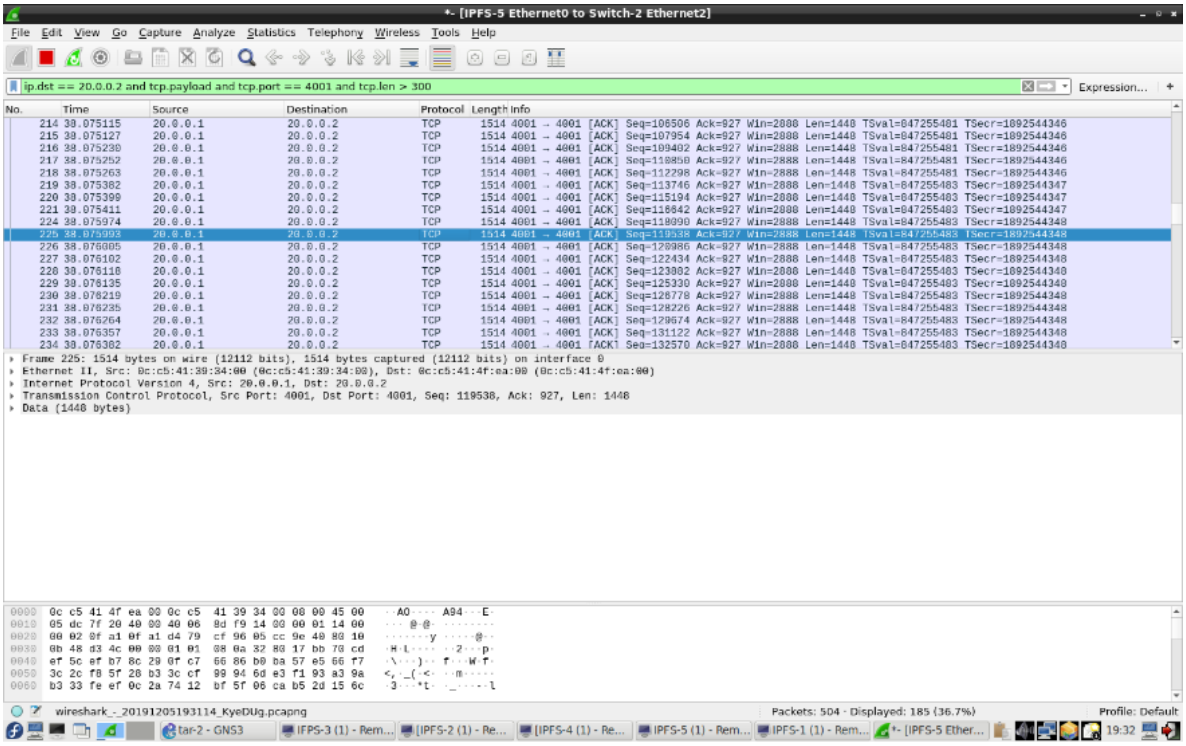


Figura 8: Exemplo 3, parte 3

```
root@osboxes:~# ipfs get QmNs6QhxfTEzH2gohnAQNwD7amzEuccVCWDz74EkyCeZY3
Saving file(s) to QmNs6QhxfTEzH2gohnAQNwD7amzEuccVCWDz74EkyCeZY3
 1.00 MiB / 1.00 MiB [=============================================================================================] 100.00% 0s
root@osboxes:~#
root@osboxes:~#
```

Figura 9: Exemplo 4, parte 1

por diferentes máquinas que contenham cópias do conteúdo em *cache*, evitando a sobrecarga de apenas uma máquina durante o processo de envio de pacotes.

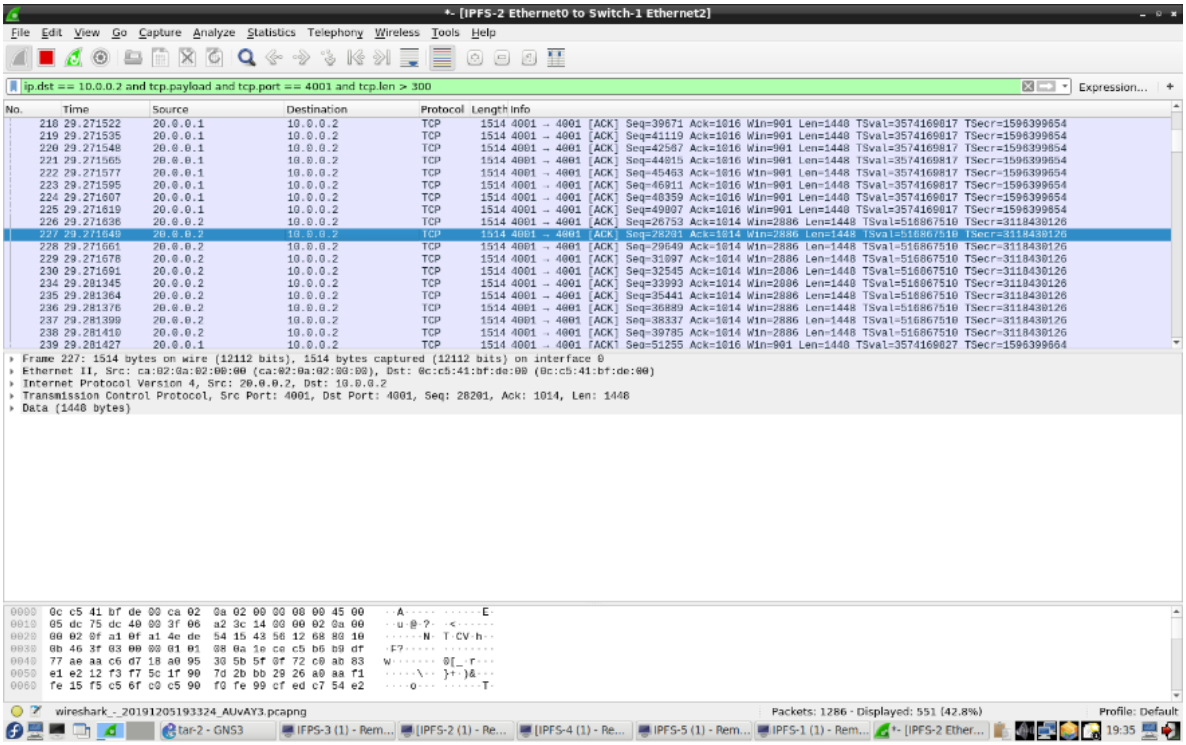


Figura 10: Exemplo 4, parte 2

## 5.5 Exemplo 5 – *Caching* "automático"

Neste exemplo, vamos fazer uso do sub-sistema `publish-subscribe` presente nesta implementação do IPFS, de forma a aumentar a redundância do conteúdo disponível na nossa rede. O nó `IPFS-1` vai subscrever ao tópico `cache-cid`, de forma a guardar na sua `cache` o conteúdo que lhe é pedido.

Começamos por verificar que determinado conteúdo não se encontra na sua *cache*, e de seguida iniciamos o `daemon` que apresentamos na secção das configurações, como ilustrado na Figura 11.

```
root@ipfs-1:~# ipfs refs local | grep QmejvEPop4D7YUadeGqYWmZxHhLc4JBUCzJJHWMzdcMe2y
root@ipfs-1:~#
root@ipfs-1:~# ./ipfs_pubsub
```

Figura 11: Exemplo 5, parte 1

De seguida, num qualquer nó terminal da nossa rede, é enviada uma mensagem `publish`, com o mesmo conteúdo que não se encontrava na *cache* do nó `IPFS-1`, mas que estaria disponível algures na nossa rede, como ilustrado na Figura 12.

Por fim, terminamos o `daemon` que estaria a correr no nó `IPFS-1`, e verificamos que o conteúdo agora se encontra em cache, como ilustrado na Figura 13.


Figura 12: Exemplo 5, parte 2

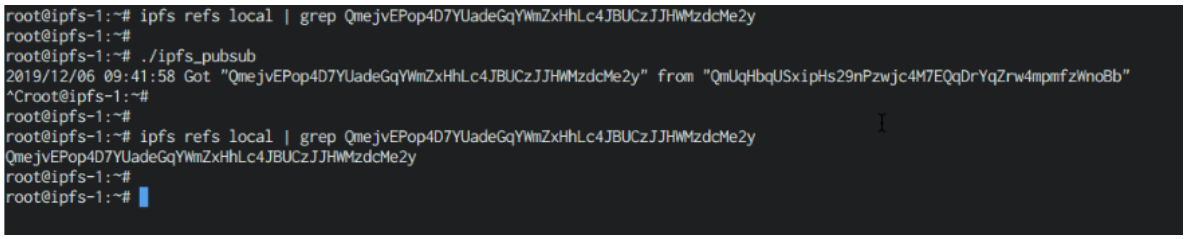


Figura 13: Exemplo 5, parte 3

# 6 Conclusão

Após uma análise dos resultados obtidos, descritos na secção anterior, concluímos que os objetivos previamente estabelecidos para este trabalho foram devidamente cumpridos. Acreditamos que foi possível transmitir um melhor entendimento das características intrínsecas das ICNs através da demonstração prática da utilização de uma rede montada à volta do IPFS, como uma tecnologia exemplificativa dos conceitos das ICNs, e dos vários casos de teste fornecidos que relatam peculiaridades do funcionamento interno do IPFS. Finalmente, foi possivel dar a conhecer o contraste entre a Internet atual e as ICNs, salientando as suas vantagens, e acentuando as fragilidades que ainda as perturbam, eventualmente limitando-as a um contexto preferêncial de utilização.

# Bibliografia

[1] T. C. Daniel Sousa Tiago Carvalho, «Information Centric Networks», *Tópicos Avançados em Redes*, pp. 1–6, 2019.

[2] «IPFS». https://ipfs.io/, Dez-2019.

[3] «GNS3». https://www.gns3.com/, Dez-2019.

[4] J. Benet, «Ipfs-content addressed, versioned, p2p file system», *arXiv preprint arXiv:1407.3561*, 2014.

[5] «Debian». https://www.debian.org/, Dez-2019.

[6] «OSBoxes». https://www.osboxes.org/, Dez-2019.

[7] «QEMU». https://www.qemu.org/, Dez-2019.

[8] «Kernel Virtual Machine». https://www.linux-kvm.org/, Dez-2019.

[9] «IPFS distributions». https://dist.ipfs.io/, Dez-2019.

[10] «Go». https://golang.org/, Dez-2019.

[11] «Systemd». https://www.freedesktop.org/wiki/Software/systemd/, Dez-2019.